Because the Office has failed to state a *prima facie* case of obviousness, the rejection should be withdrawn. Amended independent Claims 1, 11, 17, and 19 are clearly patentably distinguishable over the cited and applied references. Claims 2-10, 12-16, 18, and 20 are allowable because they depend from allowable independent claims and because of the additional limitations added by those claims. Consequently, reconsideration and allowance of Claims 1-20 is respectfully requested.

CONCLUSION

In view of the foregoing remarks, applicant submits that all of the claims in the present application, as amended, are clearly patentably distinguishable over the teachings of Brown et al., O'Connor et al., Kishimoto, and Lindholm et al. taken alone or in combination. Thus, applicant submits that this application is in condition for allowance. Reconsideration and reexamination of the application, allowance of the claims, and passing of the application to issue at an early date are solicited. If the Examiner has any remaining questions concerning this application, the Examiner is invited to contact the applicant's undersigned attorney at the number below.

Respectfully submitted,

CHRISTENSEN O'CONNOR
JOHNSON KINDNESS^PLLC

[illegible]

D.C. Peter Chu
Registration No. 41,676
Direct Dial No. 206.695.1636

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the U.S. Postal Service in a sealed envelope as first class mail with postage thereon fully prepaid addressed to Box Non-Fee Amendment, U.S. Patent and Trademark Office, P.O. Box 2327, Arlington, VA 22202, on the date specified below.

Date: July 1, 2002 Cindy S. Norton

PCC:clm

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS^PLLC
1420 Fifth Avenue, Suite 2800
Seattle, Washington 98101
206.682.8100

MSFT\17208AM.DOC

VERSION WITH MARKINGS TO SHOW CHANGES MADE JULY 1, 2002

In the Specification:

The paragraph beginning at page 1, line 16, has been amended as follows:

Therefore, computing environments have been created that are multi-threaded. A user, is thus typically able to run more than one computer program in such environments at a single time, [B] for example, both a word processing program and a spreadsheet program. Similarly, a program is in such environments usually able to run multiple threads or tasks concurrently, [B] for example, a spreadsheet program can calculate a complex formula that may take minutes to complete while concurrently permitting the user to still continue editing a spreadsheet.

The paragraph beginning at page 2, line 1, has been amended as follows:

A problem arises, however, when two threads, [B] either of the same or different programs, [B] attempt to access the same data object (which may in a non-limiting and nonexclusive sense be defined as a software component) at the same time, where exclusive access to the object is required by one or both of these threads. Such concurrent access of the same object may result in corruption of programs' data structures, ultimately causing the computer to crash. Therefore, when a given thread accesses an object, generally it is provided a [A] lock [≅] on that object, [B] for example, utilizing a lock object [B] so that other threads can only acquire limited rights to the object until the given thread is finished with using the object. For instance, to improve thread throughput, it may be desirable to implement Read-Write locks, allowing for multiple threads to read the data structure, but only one thread modifying it. Another example may be a SQL database, in which appending is a very common situation, and thus it is desirable to make it a non-blocking procedure, which is often done by implementing a complex lock structure.

The paragraph beginning at page 12, line 18, has been amended as follows:

The recyclable locking mechanism 206 associates a lock within the pool 204, such as the lock object 208, with an object, such as the object 202, using the associated variable of the object as a pointer to the lock, upon a first request by a thread, such as the thread 200, in one embodiment where the object 202 has not been previously locked by another thread. Pursuant to a second request by the thread, the mechanism 206 further deassociates the lock from the object, such that the lock is [A]recycled, [≅] that is, can be reused.

The paragraph beginning at page 15, line 12, has been amended as follows:

Referring now to FIG. 3, a flowchart of a method according to one embodiment of the invention is shown. In 300, which is an initialization phase, an associated variable of an object, such as the object 202 of FIG. 2, is reset by in one embodiment by being set to -1. The associated variable includes a set of high bits and a set of low bits, where the set of high bits followed by a zeroed set of low bits acts as a lock pointer, and the set of low bits acts as a status variable regarding whether the object is currently being used. The number of low bits <u>that</u> corresponds has to be greater or equal to the logarithm with base 2 of the maximum number of threads [thay] <u>they</u> may attempt to associate a lock with an object at the same time. Moreover, it is assumed that all locks are aligned at addresses of which all low bits are zero.

The paragraph beginning at page 16, line 22, has been amended as follows:

If the associated variable is less than the boundary value, but nonzero, this signifies that none of the high bits have been set to one, such that the set of high bits followed by a number of zero bits equal to the number of low bits (that is, the associated variable with the low bits zeroed) does not point to a lock object. Thus, the object has a spin status, meaning that a lock object is in process of being assigned to the object, such that in [308] <u>309</u> the thread waits until the lock object is pointed to by the set of high bits of the associated variable of the object, as the set of high bits is followed by a number of zero bits equal to the number of low bits. In this case, the thread shall yield control, for instance by calling Win32 function Sleep( ).

The paragraph beginning at page 18, line 8, has been amended as follows:

Once the thread has finished using the object, it decrements the associated variable of the object, in one embodiment by 1, in 314. In another embodiment, the thread accomplishes this decrementation by sending a second, unlock request to the recyclable locking mechanism, and the mechanism itself performs the decrementation. In another embodiment, the thread accomplishes the decrementation itself. In 316, it is determined whether the associated variable is [lequal] equal to a predetermined threshold, in one embodiment, -1. If this is the case, then this means that no other threads are desiring to use this object (i.e., no other objects are waiting in [310] 309 in their traversal through the flowchart of FIG. 3), and the lock object is recycled in 318 for reuse; the method then ends at 320. If this is not the case, then the method ends at 320 without recycling the lock object, since it is still being used to lock the object because the object is being used-by other threads.

In the Claims:

1. (Amended) A system comprising:

at least one thread;

a pool of locks;

at least one object that is capable of representing a resource needed by the at least one thread, [each] the at least one object having [an associated] a variable; and,

a recyclable locking mechanism [to associate] for associating a lock from the pool of locks with [an] the at least one object using the [associated] variable [of the at least one object] as a pointer [, upon a first request] when requested by [a] the at least one thread, the lock returning to the pool of locks without having to destroy the at least one object when the at least one thread no longer needs to access the resource.

3. (Amended) The system of [claim] Claim 1, wherein the [associated] variable of each of the at least one object comprises an integer.

4. (Amended) The system of [claim] Claim 1, wherein the [associated] variable of each of the at least one object comprises a set of high bits defining the pointer to a lock and a set of low bits defining a status variable.

8. (Amended) The system of [claim] Claim 7, wherein upon the set of low bits after incrementation by one being greater than 0, the [associated] variable has an in-use status by a thread such that the set of high bits points to a lock.

9. (Amended) The system of [claim] Claim 7, wherein upon the [associated] variable after incrementation by one being less than 32, the associated variable has a spin status such that the set of high bits is currently in the process of being set to a lock.

11. (Amended) A method comprising:

asserting an instruction by a thread to lock an object;

increasing [an associated] a variable of the object, the [associated] variable having a set of high bits for representing a pointer to a lock and a set of low bits for representing a lock status; [and,]

determining whether the [associated] variable is greater than a boundary value so as to allocate the lock; and

recycling the lock by returning the lock to a pool of locks when the thread no longer needs the object regardless of whether the object persists after the lock returns to the pool of locks.

12. (Amended) The method of [claim] <u>Claim</u> 11, further comprising initially setting the [associated] variable of the object to -1.

13. (Amended) The method of [claim] <u>Claim</u> 11, further comprising upon determining that the [associated] variable is less than the boundary value, waiting until the [associated] variable is greater than the boundary value.

14. (Amended) The method of [claim] <u>Claim</u> 11, further comprising upon determining that the [associated] variable is greater than the boundary value, using the set of high bits of the [associated] variable as a pointer to a lock for the object.

15. (Amended) The method of [claim] <u>Claim</u> 14, further comprising:

decrementing the [associated] variable of the object; and, determining whether the [associated] variable is less than a minimum threshold.

16. (Amended) The method of [claim] <u>Claim</u> 15, upon determining that the [associated] variable is less than the minimum threshold, recycling the lock.

17. (Amended) A computer comprising:

a processor;

a computer-readable medium; and,

a recyclable locking mechanism program executed by the processor from the medium to associate a lock with an object using [an associated] <u>a</u> variable of the object as a pointer <u>when requested by a thread, the lock being capable of returning to a pool of locks without having to destroy the object when the object is no longer needed by the thread</u>.

18. (Amended) The computer of [claim] Claim 17, wherein the [associated] variable of the object comprises a set of high bits defining the pointer to a lock and a set of low bits defining a status variable.

19. (Amended) A computer-readable medium having a recyclable locking mechanism program stored thereon for execution on a computer to associate a lock with an object using [an associated] a variable of the object as a pointer when requested by a thread, the lock being capable of returning to a pool of locks without having to destroy the object when the object is no longer needed by the thread.

20. (Amended) The computer-readable medium of [claim] Claim 19, wherein the [associated] variable of the object comprises a set of high bits defining the pointer to a lock and a set of low bits defining a status variable.